

# Máster Avanzado de Programación en Python para Hacking, BigData y Machine Learning

Programación Avanzada Python

# LECCIÓN 05

## Trabajando con colecciones

# ÍNDICE

Introducción

Objetivos

Cadenas

Listas

Conjuntos

Diccionarios

# INTRODUCCIÓN

En esta cuarta lección vamos a estudiar como trabajar distintos tipos de colecciones en Python. Veremos desde los más simples como son las cadenas, pasando por las más utilizadas (listas) y seguido de los conjuntos y diccionarios muy útiles también para determinados tipos de problemas.

# OBJETIVOS

Al finalizar esta lección serás capaz de:

- 1 Las principales tipos de datos de colecciones.
- 2 Trabajar con cadenas, listas, conjuntos y diccionarios.
- 3 Diferenciar cada una de las colecciones.
- 4 Buscar otras funcionales adicionales a estos tipos de datos

# Cadenas

Una cadena puede definirse en Python entre comillas simples y dobles. Una vez declarada podemos imprimirla por pantalla con la función print

```
print("string")  
  
string
```

Si la cadena son sentencias que tienen más de una línea, podemos declararla con 3 comillas dobles o simples:

```
var = """hello  
it is a string  
yes"""
```

## Indexación de cadenas

El índice de la cadena comienza desde cero y se puede acceder a cada carácter especificando su valor de índice.

Sintaxis: `nombre_de_la_cadena_variable [valor_del_índice]`

-6	-5	-4	-3	-2	-1
s	t	r	i	n	g
0	1	2	3	4	5



### Indexación positiva

```
var = "strong"  
print(var[0])  
print(var[1])  
print(var[5])
```

```
s  
t  
g
```

### Indexación negativa

```
print(var[-1])  
print(var[-6])
```

```
g  
s
```

### Slicing

```
print(var[2:5])
```

```
ron
```

### Recorriendo cadenas

```
for i in "string":  
    print(i)
```

Es posible comprobar la existencia de una cadena en particular con la palabra clave 'in' y retornar un valor booleano verdadero si está presente.

```
sentence = "Python is simple to learn"  
print("simple" in sentence)
```

```
True
```

```
if "simple" in sentence:  
    print("The word found")
```

```
The word found
```

## Operadores para cadenas

Los principales operadores utilizados para las cadenas son + y \*.

```
a = "string.."
print(a * 3)

string..string..string..
```

```
>>> "hola " + "mundo"
'hola mundo'
>>> 
```

## Funciones de cadena

`chr ()` - convierte un entero en carácter o su correspondiente valor ASCII

<code>chr(97)</code>	<code>chr(33)</code>
'a'	'!'

`ord ()` - realiza la operación inversa a `chr ()` y convierte el correspondiente valor entero o ASCII

<code>ord("!")</code>	<code>ord("a")</code>
33	97

## Funciones de cadena

`str ()` - devuelve el parámetro especificado como una cadena. Si se comprueba el tipo de su valor devuelto, será una cadena independientemente del tipo del parámetro.

```
num = str(43)
print(num)
print(type(num))
```

```
43
<class 'str'>
```

`replace ()` - para sustituir un carácter por otro.

```
var.replace('c','r')
```

```
'rat'
```

# Listas

La lista es un conjunto de elementos almacenados de forma ordenada.

```
pet_list = ['dog', 'cat', 'rabbit']  
  
print(pet_list[2])  
  
rabbit
```

Sintaxis:

Nombre\_de\_la\_lista = [elemento1, elemento2..., elemento n]

No hay límite para el número de elementos y éstos pueden ser de diferentes tipos de datos.

## Métodos de la lista

`append()` - Se utiliza para añadir un solo elemento al final de la lista. Puede añadir fácilmente una cadena de números, tuplas y listas.

```
list = []  
  
list.append(1)  
list.append(2)  
print(list)  
  
[1, 2]
```

```
list.append(('hello', 'hai'))  
  
print(list)  
  
[1, 2, [3, 4], ('hello', 'hai')]
```



`insert()` - La función `insert` tiene el mismo propósito que `append()`, añadir elementos a la lista. Esta función puede añadir el elemento en la posición deseada según los parámetros especificados.

```
num_list  
  
[0, 1, 2, 3, 4]  
num_list.insert(0, 'a')  
  
print(num_list)  
  
['a', 0, 1, 2, 3, 4]
```

```
num_list.insert(5, 'b')  
print(num_list)  
  
['a', 0, 1, 2, 3, 'b', 4]
```

`extend()` - Esta función es la misma que `insert` y `append` utilizada para añadir elementos a la lista. Añade elementos al final de la lista como la función `append` y múltiples elementos como una lista de elementos.

```
list = [0, 1, 2, 3, 4]
list.extend([5,6,7,8,9,10])
print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

`extend()` - Esta función es la misma que `insert` y `append` utilizada para añadir elementos a la lista. Añade elementos al final de la lista como la función `append` y múltiples elementos como una lista de elementos.

```
list = [0, 1, 2, 3, 4]
list.extend([5,6,7,8,9,10])
print(list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

`remove()` - La función `remove` elimina los elementos de la lista. Elimina un elemento a la vez especificando dicho elemento por parámetro.

```
list.remove(5)

print(list)

[0, 1, 2, 3, 4, 6, 7, 8, 9, 10]
```

`pop()` - La función `pop` se utiliza para eliminar los elementos de la lista. La función puede ser utilizada con o sin parámetro. Por defecto elimina el elemento del final de la lista.

```
list = [0, 1, 2, 3, 4, 5, 6]
list.pop()
print(list)

[0, 1, 2, 3, 4, 5]
```

`clear()` - Elimina todos los elementos de la lista.

```
list = [0, 1, 2, 3, 4,5,6]
list.clear()
print(list)
```

```
[]
```

`index()` - Devuelve el índice del elemento especificado en el parámetro.

```
list = [1, 2, 3, 4,5,6]
index = list.index(2)
print(index)
```

```
1
```

`count()` - Devuelve el número de veces que se repite el elemento pasado por parámetro.

```
list = [1,2,3,3,4,3,5,3]
count = list.count(3)
print(count)
```

```
4
```

`sort()` - Ordena los elementos de la lista en orden ascendente por defecto.

```
list = [3,1,5,4,6,2]
list.sort()
print(list)
```

```
[1, 2, 3, 4, 5, 6]
```

`reverse()` - Invierte el orden de la lista.

```
list = [1, 2, 3, 4, 5, 6]
list.reverse()
print(list)
```

```
[6, 5, 4, 3, 2, 1]
```

`copy()` - Devuelve una copia de una lista y no utiliza ningún parámetro.

```
list = [1, 22, 333, 4444]
list1 = list.copy()
print(list, list1)
```

```
[1, 22, 333, 4444] [1, 22, 333, 4444]
```



`sum()` - Devuelve la suma de todos los elementos de la lista

```
list = [1, 2, 3, 4,5,6]
sum = sum(list)
print(sum)
```

21

`min()` - Devuelve el elemento mínimo o más pequeño de la lista

```
list = [1, 2, 3, 4,5,6]
sum = min(list)
print(sum)
```

1

`max()` - Devuelve el elemento máximo o más grande de la lista.

```
list = [1, 2, 3, 4,5,6]
sum = max(list)
print(sum)
```

6

# Conjuntos

Los conjuntos es una estructura de datos, que recogen los elementos dentro de llaves '{}'. La principal característica de los conjuntos es que son desordenados y la indexación no funciona para este tipo de datos.

Sintaxis:

nombre\_conjunto = {elemento 1, elemento 2... elemento n}

```
country = {"India", "USA", "UAE"}
```

`add()` - Los conjuntos pueden añadir elementos utilizando la función `add`. Se añade un único elemento en cualquier lugar

```
set = {1,2,4}
set.add(3)
print(set)
```

```
{1, 2, 3, 4}
```

`update()` - Con esta función se pueden añadir múltiples elementos en el parámetro y deben ser tuplas, listas o cadenas.

```
set = {1,2,4}
set.update([5,6],{7,8})
print(set)
```

```
{1, 2, 4, 5, 6, 7, 8}
```

`discard()` - Se utiliza para eliminar un elemento del conjunto. Elimina un solo elemento a la vez.

```
set = {1,2,4}
set.discard(4)
print(set)
```

```
{1, 2}
```

`union()` - La unión de dos conjuntos es la combinación de ambos conjuntos.

```
set1 = {2,3,4}
set2 = {5,6,7}
print(set1 | set2)
```

```
{2, 3, 4, 5, 6, 7}
```

`intersection()` - La intersección de dos conjuntos contiene los elementos comunes que está presente en ambas listas.

```
set1 = {2,3,4}
set2 = {5,3,2}
print(set1 & set2)
```

```
{2, 3}
```

`difference()` - La diferencia entre dos conjuntos significa, conjunto1 - conjunto2, contiene los elementos del conjunto1 pero no del conjunto2.

```
set1 = {2,3,4}
set2 = {5,3,2}
print(set1 - set2)
```

```
{4}
```

# Diccionarios



Los diccionarios son otro tipo de estructura de datos. Aquí cada elemento del diccionario tiene 2 parámetros: clave y valor.

Sintaxis:

Nombre\_diccionario: { clave1: valor1, clave2:valor2, ..... }

```
emails = { "xyz" : "xyz@gmail.com",  
           "qas" : "qas@yahoo.com",  
           "qwe" : "qwe@gmail.com" }
```

`get()` - Se utiliza para acceder al valor con la clave como argumento

```
dict = {'name': 'abc',  
        'age': 44}  
  
print(dict.get('name'))  
  
abc
```

`pop()` - Se utiliza para eliminar un valor del diccionario dando la clave como parámetro.

```
dict = {'name': 'abc',  
        'age': 44}  
  
dict.pop('name')  
print(dict)  
  
{ 'age': 44 }
```

`popitem()` - Se utiliza para eliminar un par clave-valor del diccionario.

```
dict = {'name': 'abc',  
        'age': 44}  
  
dict.popitem()  
print(dict)  
  
{'name': 'abc'}
```

`clear()` - Elimina todos los elementos del diccionario.

```
dict = {'name': 'abc',  
        'age': 44}  
  
dict.clear()  
print(dict)  
  
{}
```

## CONCLUSIONES

1

El índice de la **cadena** comienza desde cero y se puede acceder a cada carácter especificando su valor de índice.

2

La **lista** es un conjunto de elementos almacenados de forma ordenada.

3

La principal característica de los **conjuntos** es que son desordenados y la indexación no funciona para este tipo de datos.



MUCHAS GRACIAS POR SU ATENCIÓN



[rsanchezi@grupomainjobs.com](mailto:rsanchezi@grupomainjobs.com)



Rubén Sánchez Iruela

[linkedin.com/in/ruben-sanchez-iruela-8156799a](https://www.linkedin.com/in/ruben-sanchez-iruela-8156799a)



[twitter.com/eiposgrados](https://twitter.com/eiposgrados)



[facebook.com/eiposgrados](https://facebook.com/eiposgrados)



[instagram.com/eiposgrados](https://instagram.com/eiposgrados)